Static Testing and Dynamic Testing

1. **Static Testing**
    1. Testing without executing the Software Code
    2. Tests Code and Documents
    3. Static Testing is nothing but a technique of Verification Testing
    4. Reviews, Walkthroughs and Inspections are part of Static Testing

    Static Testing Techniques
    Static Analysis - Analyze the code written by the developers with the help of Tools

    Reviews - Reviews are performed to find the mistakes/ambiguities in the documents and code.
    Reviews can be further categorized into:
    1. **Walkthrough** - Informal review meeting conducted with an intention of gathering feedback from Project Team on documents (Guided by Author of document)
    2. **Inspection** - Formal review meeting where different roles like Moderator, Authors, Inspectors, Readers, Scribe etc. are created and conducted in a process oriented way on documents (Schedule, Organizing, Email Invitation and After Meeting mail etc.)
    3. **Code Review** - Examining the source code of the Software for identifying bugs and then removing them from the source code
    4. **Pair Programming** - Two people work on the same code by sitting at a single workstation with an intention of carrying out the coding and review process parallel

2. **Dynamic Testing**
    1. Testing by executing the Software Code
    2. Dynamic Testing is nothing but a technique of Validation Testing
    3. Unit Testing, Integration Testing, System Testing and User Acceptance Testing are part of Dynamic Testing

Levels of Software Testing

There are four levels of Software Testing which can be performed across the Software Development Life Cycle

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

1. **Unit Testing**
    1. A unit is the smallest testable part of Software
    2. Unit Testing falls under **White Box** Testing category as we are testing the internal logic of the program
    3. Unit Testing is done by the Developers

2. **Integration Testing**
    1. Individual Units are integrated logically and tested as a group
    2. The communication between the different Units integrated is working properly and all Units are working together properly will be tested
    3. Integration Testing also falls under **White Box Testing** category as we are testing the internal logic of the program
    4. Integration Testing is done by the Developers
    5. Approaches followed in Integration Testing:
        - Big Bang Approach - All the units (modules) will be integrated at a go after every module is ready and the resulted functionality will be tested.
        - Top Down Approach - Higher level modules are tested first, followed by the low level modules integration to their higher level modules
            1. stubs (temporary programs) are used if any modules are not ready
        - Bottom Up Approach - Reverse of Top Down approach
            1. drivers (temporary programs) are used if any modules are not ready
        - Sandwich Approach (Hybrid)
            1. Top Down + Bottom Up

3. **System Testing** - Testing performed on the complete system and it is performed by Software Testers. System testing falls under Black Box Testing category.

    As part of System Testing, we generally perform
    1. **Function Testing**
    2. **Non-Functional Testing**
    3. **UI Testing**
    4. **Usability Testing**

4. **User Acceptance Testing**
    - User Acceptance Testing is performed keeping end user in mind and for building confidence before releasing the product into the market
    - UAT Testing (Alpha and Beta Testings) is performed after System Testing is passed and when the build is stable
    - Types of User Acceptance Testing
        1. **Alpha Testing**
            1. Alpha Testing as mentioned is one of the User Acceptance Testing Types
            2. Alpha Testing is performed by In House Testers, potential customer or users before providing the product to the end users or public
            3. Alpha Testing is performed by Project Team's Testers, potential customers or users at the Developers site in testing environment
            4. Alpha Testing is performed before Beta Testing
        2. **Beta Testing**
            1. Beta Testing as mentioned is one of the User Acceptance Testing Types
            2. Beta Testing is performed by real users of the Software in real environments
            3. Defects and User Feedback will be collected as part of Beta Testing

Principles Of Testing

1. **Testing shows the presence of bugs**
    1. Testing cannot prove that the application is defect free
    2. Testing reveals the defects in the Software, which doesn't mean that the application will become defect free because of testing i.e., The goal of the testing is to find as many defects as possible instead of making the application defect free.
2. **Exhaustive testing is impossible**
    1. Testing with all valid and invalid combinations of input may not be possible
    2. Example: Testing an input field which can intake numbers from 1 to 1 Lack
    3. We test them based on risk and priorities
3. **Early Testing**
    1. Testing should start from the early stages of SDLC
    2. Defects found in early stages of the SDLC, are cheaper to fix than those found in the later stages
        1. Defects identified during earlier stages cost less
        2. Defects identified after release are expensive
4. **Testing is Context Dependent**
    1. Testing of different domains has to be done differently
    2. The risk associated with different application domains is different
        1. Example: Testing Banking application is different for Schooling Assignments application
5. **Defect Clustering**
    1. The majority of the defects come from a small number of modules
    2. The defects won't be equally distributed across the application
6. **Pesticide Paradox**
    1. Pesticide Paradox principle says that if the same set of test cases are executed again and again over the period of time, these tests won't capable enough to identify new defects in the Software
    2. In order to overcome Pesticide Paradox, we need to review and update the test cases regularly
7. **Absence of Error - fallacy**
    1. 99% of bug-free software may still be unusable, if wrong requirements were incorporated into the software and the software is not addressing the business needs.

Software Test Life Cycle [STLC]

It is a process that we follow to test the software

1. Requirement Study
2. Write a test plan
3. Write a test scenario
4. Write a test case
5. Traceability Matrix: It is a document that maps between requirements number and test case id. This is written in order to check whether every requirement has at least one test case associated with it.
6. Execution of test cases
7. Defect tracking
8. Test execution report or quality report

Design Techniques to Write The Test Cases

Boundary Value Analysis

In this technique first we will consider the boundary values to write the test cases.

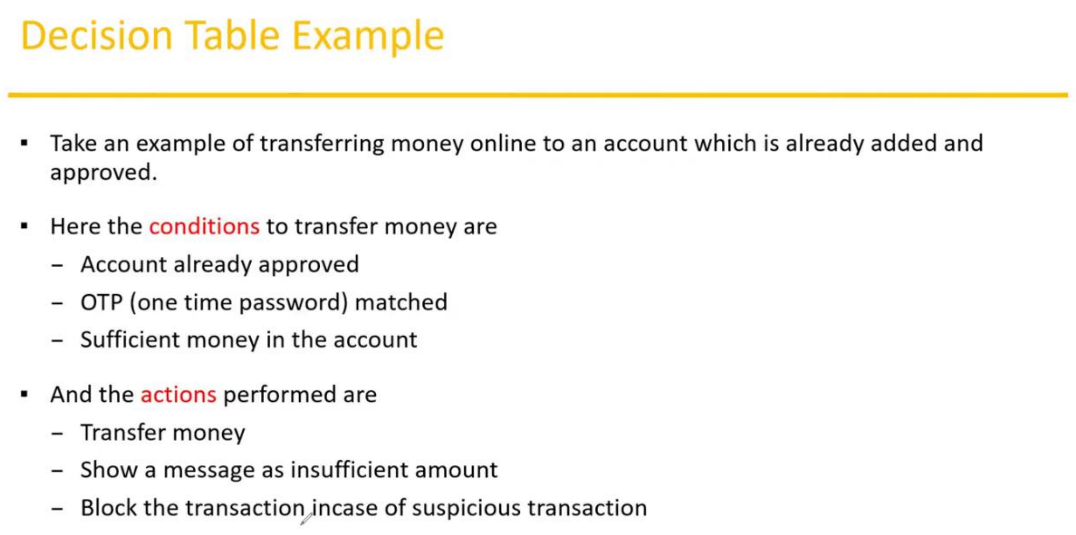
Then we will consider the nominal value and we will use one less than minimum and one more than maximum to write test cases.

Equivalence Class Partitioning

In this technique the input data is divided into different equivalence data classes. This method is typically used to reduce the total number of test cases into a finite number of testable test cases.

Decision Table

The decision table design technique is a systematic approach to creating decision tables that accurately capture all possible combinations of conditions and corresponding actions. It is particularly useful for representing complex decision-making scenarios with multiple inputs and outputs. For example



Decision Table Example

- Take an example of transferring money online to an account which is already added and approved.
- Here the conditions to transfer money are
  - Account already approved
  - OTP (one time password) matched
  - Sufficient money in the account
- And the actions performed are
  - Transfer money
  - Show a message as insufficient amount
  - Block the transaction incase of suspicious transaction

State Transition

In State Transition technique changes in input conditions change the state of the application. This testing technique allows the tester to test the behavior of an AUT.

In State Transition technique, the testing team provides positive as well as negative input test values for evaluating the system behavior.